Simulation of Pool Testing to Identify Patients With Coronavirus Disease 2019 Under Conditions of Limited Test Availability

Shiny App Website

https://abrahamliu7.shinyapps.io/COVID_19_Simulation_Model/

Introduction

The novel coronavirus disease (COVID-19), resulting from the serious acute respiratory syndrome (SARS-CoV-2) virus is a highly contagious respiratory disease, first originating in the United States in January 2020. With the widespread of COVID-19 pandemic, efforts have been made for efficient testing measures. The patients who are suspected to have COVID-19 go through a real-time reverse transcriptase–polymerase chain reaction - RT-PCR testing. (Source: https://jamanetwork.com/journals/jamanetworkopen/fullarticle/2767513?resultClick=3). The aim of RT-PCR is to check for RNA detection of SARS-CoV-2. The increased request for the SARS-CoV-2 RT-PCR tests has led to a decreased supply of testing reagents. Hence, researchers are examining pooled testing measures from multiple patients. With the pooled testing, if the COVID-19 tests results are negative, then it can be concluded that all sampled individuals do not have COVID-19. However, if the pooled test results are positive, then the sample from every individual is seperately examined/tested. Pooled testing works best under limited testing conditions. However, if we examine sensitivity of the test (a measure which reflects patients with true positive tests), and observe a sensitivity under 100%, this results in the likelihood of false negatives. Therefore, it is important to examine a false negative risk by looking at three different criteria: the prevalence of COVID-19, the sensitivity of the COVID-19 tests, and the pool size of the patient.

COVID-19 Pooled Testing Analysis

In the original model for pooled testing (Source: The JAMA Network), every individual in a group will mix their blood in one sample. In order to stimulate this, we assume that the status of positive and negative results are indeed true positives. Thus, we label negative results as 1, and positive results as 0. The product of all values will be the mixed blood result. If the mixed blood result contains solely negative samples, this will result in 1 (based on prior notation). Otherwise, the result will be 0.

Our shiny app for COVID-19 Pooled Testing provides users with a platform to implement pooled testing strategies from patients with suspected COVID-19 disease. The app stimulates running pooled tests. The number of positive and negative cases are gathered to create a population, as the user will be able to sample a random group sample size n. In addition, the user will specify the sensitivity (true positive rate) and specificity (true negative rate) levels for any state of their choice. Based on the values the user specifies, the population will be sampled without replacement and tested for either true negative or true positive pooled results. The stimulated model will return either a positive or negative result, which will determine the number of tests that need to be run.

Datasets

We have used Kaggle to get our datasets for "COVID-19 in USA." These datasets are originally adapted from the "New York Times" and the "COVID-19 Tracking Project." The information about COVID-19 and simulation with pooled testing is adapted from the "JAMA Network." The website links are also included

below.

Website Sources: - https://jamanetwork.com/journals/jamanetworkopen/fullarticle/2767513?resultClick=3 - https://www.kaggle.com/sudalairajkumar/covid19-in-usa?select=us_states_covid19_daily.csv

Corresponding csv File Used for COVID-19 Data: - us_states_covid19_daily.csv

We have also used the first 99 words of the Bee Movie script as a fun little joke while the shiny app simulates the data (since the calculations will take a while if the population size is large). The source code of the script is found here: https://gist.github.com/ElliotGluck/64b0b814293c09999f765e265aaa2ba1

Corresponding txt file used for the Bee Movie Script: - progresscheck.txt

Data Generating Process

Our dataset comes from Kaggle (https://www.kaggle.com/sudalairajkumar/covid19-in-usa?select=us_states_covid19_daily.csv), which includes data about COVID-19 in all fifty US States. From Kaggle, we have downloaded the csv file "us_states_covid19_daily.csv," containing daily-level data about coronavirus tests in the United States (i.e. following month/date/year format for data collection). The data generation and cleansing process is performed prior to running the Shiny App.

Data Origination and Legality

Kaggle (https://www.kaggle.com/) provides users with an array of data for projects and analysis. The website for our Shiny App, (https://www.kaggle.com/sudalairajkumar/covid19-in-usa?select=us_states_covid19_daily.csv), provides COVID-19 data accessible to the public, with appropriate sources credited ("COVID-19 Tracking Project" and "New York Times"). The data generated follows appropriate terms of use.

Packages

Require the following libraries: `shiny`, and `shinythemes` for generating interactive dashboards; `tidyverse` for cleaning data and manipulating datasets; `plotly` for creating interactive visuals, `lubridate` for dates manipulation, and `docstring` for documentation

In-App

Functions

`server(input, output)`

Defines the server logic ```r population <- function(df = df, state = state, date = date, prev = 0.01, pop = 10000) { #' Returns vector of 1's and 0's to simulate a population for COVID-19 Status #' @param df dataframe, default dataframe with unspecified dimensions #' @param state character, default length 1 character type #' @param date numeric, default length 1 numeric type #' @param prev numeric, default length 1 numeric type #' @param pop numeric, default length 1 numeric type #' @return the population vector with length of size population

```
  # if user decided to use the COVID-19 Dataset
  if (input$rd == 1) {
    positives <- df$positive[df$date == date & df$state == state] #grab number
    negatives <- df$negative[df$date == date & df$state == state] #grab number
  }
  #if user decided to simulate with using a prevalence parameter
  else {
```

```
    negatives <- rbinom(1, pop, 1-prev)
    positives <- pop - negatives
  }
  return(c(rep(1, negatives), rep(0, positives)))
  #return vector with positive cases having values of 0, negatives with 1
}

The population function is to create a vector with the size of the total amoun
```r
pooled_test <- function(sample=c(), se=0, sp=0, size = 2) {
    #' Returns test result status for a particular pooled group
    #' @param sample vector, default length size numeric type
    #' @param se numeric, default value 0
    #' @param sp numeric, default value 0
    #' @param size numeric, default value 2
    #' @return value 1 or value size + 1 depending on sample and se + sp.
    pool <- prod(sample) #simulate pooling everyone into one sample
    if (pool == 1) { #if everyone within the test are truly negative
      test_prob <- runif(1,0,1)
      if (test_prob < sp) {return(1)} #true negative pooled result
      else {return(size + 1)} #false positive result
    }
    else {# if there exists at least one person that truly has COVID
      test_prob <- runif(1,0,1)
      if (test_prob < se) {return(size + 1)} #true positive pooled result
      else {return(1)} #false negative pooled result
    }
  }
```

The pooled_test function is to simulate a test for a group's pooled sample. To simulate the idea of pooling everything into one sample, all the values within the sample group vector have been multiplied together. If the resulting product is 1, then everyone in the group is a true negative. Otherwise, there exists at least one true positive in the group. Afterwards, a random number between 0 and 1 is sampled, and it is compared to the sensitivity or specificity level depending on if the product value is 0 or 1. From the if statements, either 1 or size + 1 is returned. ```r pooled_sample <- function(df=df,date=date, state=state, prev = 0.01, pop = 10000, se=0,sp=0, size=2) { #' Returns the total number of tests needed in a random simulation of the pooled-sample test model #' @param df dataframe, default dataframe with unspecified dimensions #' @param date vector, default length 1 numeric type #' @param state vector, default length 1 character type #' @param prev numeric, default length 1 numeric type #' @param pop numeric, default length 1 numeric type #' @param se numeric, default value 0 #' @param sp numeric, default value 0 #' @param size numeric, default value 2 #' @return numeric value of at least 1

```
  # make our population vector
  pop <- population(df = df, state = state, date = date, prev = prev, pop = po
  #initialize our test counter
  total_tests <- 0
  for (i in 1:ceiling(length(pop)/size)) {#iterate for the amount of groups po
    #gather indices for randomly selecting a group with "size" amount of peopl
    sample_indices <- sample(x = 1:length(pop), size = min(size,length(pop)),
    #run the pooled_test function with the newly created group, append the val
    total_tests <- total_tests + pooled_test(pop[sample_indices], se, sp, size
    #update our population by removing the pooled samples using sample_indices
    pop <- pop[-sample_indices]
  }
  return(total_tests)
}

This function either simulates an entire test run for the entire day, or simul
Due to the sensitivity and specificity parameters, the results of this model i
```r
```

```
binary_search <-function(df=df,date=date, state=state, prev=0.01, pop=10000,
                         se=0, sp=0) {
    #' Returns the estimated group size value that gives the lowest average te
    #' @param df dataframe, default dataframe with unspecified dimensions
    #' @param date vector, default length 1 numeric type
    #' @param state vector, default length 1 character type
    #' @param prev numeric, default length 1 numeric type
    #' @param pop numeric, default length 1 numeric type
    #' @param se numeric, default value 0
    #' @param sp numeric, default value 0
    #' @return numeric value of at least 2
    #min group size possible is 2
    min_k <- 2
    #max group size bound the total population *3/4 rounded up
    #max group size possible would only have 2 possible values: 1 and populati
    #These 2 extremes would skew the average due to sensitivity and specificit
    #Thus we pick a bound 3/4th of the total population for a better predictio
    if (input$rd == 1) {
      max_k <- ceiling(df$totalTestResults[df$date == date & df$state == state
    }
    else {
      max_k <- ceiling(pop*3/4)
    }
    #find empirical average of tests needed for max group size to compare with
    test_results <- mean(sapply(1:100, function(x) {
      pooled_sample(df=df, date=date, state=state, prev = prev, pop = pop,
                    se=se, sp=sp, size = max_k)
    }))
    #while loop when there is more than 1 number in our search range
    while (max_k - min_k > 1) {
      #grab a number in the middle of the search range
      new_k <- ceiling((max_k+min_k)/2)
      #get an empirical average of tests under new k group
      new_results <- mean(sapply(1:100, function(x) {
        pooled_sample(df=df, date=date, state=state, prev = prev, pop = pop,
                      se=se, sp=sp, size = new_k)
      }))
      #if this new average is greater than (or equal to) the test results
      if (new_results >= test_results) {
        #minimum is to the right. Change left bound for next iteration
        min_k <- new_k
      }
      #if new average is less than the test results
      else {
        #minimum is to the left. Change right bound for next iteration.
        max_k <- new_k
        #update test_results with new_results as the new value to be compared
        test_results <- new_results
      }
    }
    return(ceiling((max_k+min_k)/2))#return median rounded up
  }
```

This function tries to estimate the group size value that gives the lowest average tests conducted through an improvised version of binary search. First, the upper and lower bounds of where the minimum group size value can be found is defined. The lower bound will always be 2 since a group needs to be at least size 2, or else we wouldn't be doing a pooled sample test. The upper bound is 3/4th of the total population size, rounded up. The reason being is that if the group size is too high, then the average would be skewed due to the limited extreme values that are possible. For example, if we do a group with the max population size, the

only 2 outcomes are 1 and size + 1, and the variability in simulations could result in a average that is among the lowest available. This would then give incorrect judgements for the binary search in the future, resulting in wrong results. Thus, we limit the upper bound and set it to 3/4 of the total population rounded up.

After determining our bounds, we then calculate the average amount of test results for one end (which would be running our pooled_sample function a hundred times and averaging the number). Because trying to calculate the average with group size 2 is infeasible for high population sizes (potentially resulting in millions or maybe billions of iterations), we opt with calculating the average for the upper bound. A while loop is then conducted under the condition that the difference between the 2 bounds is greater than 1. The average test results of the median value rounded up is compared to the average test results of the upper bound within each iteration. If the results of the median is greater than our equal to the results of the upper bound, then this suggests that the group size value which gives the minimum average is to the right of the median. We thus change our lower bound to be this median value and move on to the next iteration. If the results of the median is less than the results of the upper bound, this suggests that the group size value which gives the minimum value is to the left of the median. We thus change our upper bound to this median, and use the median's average test results for future comparison. We then move on to the next iteration.

Because of the randomness of the pooled test model, this function is also inherently random, since there is no way to feasibly calculate the minimum average test results when the potential group sizes become large (and thus use the proper binary search function by comparing the absolute minimum value). Thus, in an attempt to circumvent this, we have also tried running this function multiple times to find the distribution of estimated group sizes that give the minimum average test results (this time 300 runs of pooled_sample since we have less things to check, so we want a more stable average), and then picking the test result that gives the lowest average. If the user has specified in using simulations with a prevalence parameter, a formula (to calculate the optimal group size) from the article we based our model off on is stated for a comparison between the estimated optimal group size versus the article's optimal group size.